Amendments to the Claims:

This listing of claims replaces all prior versions and listings of claims in the application:

Listing of Claims:

1. (Currently Amended) In a computer program, a method for managing ~~maintaining~~ dependencies among a first set of objects each having a value, the first set of objects including ~~an object A and an~~ a first object [[B]] and one or more second objects depending directly or indirectly on the first object, the one or more second objects including at least one observer-only second object, the method for maintaining dependencies comprising:

~~when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;~~

when the value of the first object [[B]] changes, invalidating the ~~dependents of object B and all of their further dependents~~ one or more second objects, including severing dependencies among the ~~dependents of object B and all of their further dependents~~ one or more second objects, the value of each second object only being recomputed when the respective second object is queried for a value; and

causing each invalidated observer-only second object to recompute its value by querying the values of [[the]] a second set of objects from which the observer-only second object depends, wherein querying the values of the second set of objects causes invalidated objects in the second set of objects to be recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

2. (Currently Amended) The method of claim 1, further comprising:

providing the first object [[B]] in [[the]] a construction of each second object [[A]], wherein the value of each second object [[A]] is a function of the value of the first object [[B]] that was provided in the construction of the respective second object [[A]].

3. (Currently Amended) The method of claim 1, further comprising:

providing in the first object [[B]] a method that adds a requester owned by a second object [[A]] to a dependents list for the first object [[B]], the dependents list identifying all second objects whose value is a function of the value of the first object [[B]].

4. (Currently Amended) The method of claim 3, wherein the dependents lists for all objects in the first set of objects collectively define a directed, acyclic dependency graph.

5. (Currently Amended) The method of claim 1, further comprising:

when [[an]] a second object is invalidated ~~marked as dirty~~, breaking any dependency relationships the ~~marked~~ invalidated second object may have had; and

when the value of an invalidated second object is recomputed, identifying the objects on which the recomputed value is actually dependent and identifying the recomputed second object as dependent only on the identified objects.

6. (Currently Amended) The method of claim 1, wherein the first set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid.

7-10. (Canceled)

11. (Currently Amended) A method for changing objects having values defining state of a computer program application, comprising:

receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

registering the change with a transaction;

dirtying all objects dependent on the changed object;

severing dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueuing the leaf object for synchronization after the transaction is committed, wherein leaf object synchronization includes using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

12. (Canceled)

13. (Previously Presented) The method of claim 11, wherein leaf object synchronization comprises:

recomputing a value for each object marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

14. (Canceled)

15. (Currently Amended) A method for changing objects defining state of a computer program application, comprising:

creating a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

for each change registered, traversing a dependency graph from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph, wherein the values for objects marked as dirty are recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

16. (Original) The method of claim 15, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

17. (Original) The method of claim 15, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

18. (Original) The method of claim 17, wherein:

the leaf objects are coupled directly to the user interface.

19-27. (Canceled)

28. (Currently Amended) A system for managing ~~maintaining~~ dependencies among a first set of objects in a computer program, each object having a value, the first set of objects including ~~an object A and an~~ a first object [[B]] and one or more second objects depending directly or indirectly on the first object, the one or more second objects including at least one observer-only second object, the system comprising:

~~means for recomputing the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;~~

means for recomputing the value of the first object [[B]], wherein when the value of the first object [[B]] changes, ~~invalidating the dependents of object B and all of their further dependents~~ the one or more second objects are invalidated, including severing dependencies among the ~~dependents of object B and all of their further dependents~~ one or more second objects, the value of each second object only being recomputed when the respective second object is queried for a value; and

means for causing each invalidated observer-only second object to recompute its value by querying the values of [[the]] a second set of objects from which the observer-only second object depends, wherein querying the values of the second set of objects causes invalidated objects in the second set of objects to be recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

29-30. (Canceled)

31. (Currently Amended) A system for changing objects having values defining state of a computer program application, comprising:

means for receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

means for registering the change with a transaction;

means for dirtying all objects dependent on the changed object; and

means for severing dependencies from the changed object and all of its direct and indirect dependent objects; wherein

whenever a leaf object is encountered as a dependent object, the leaf object is enqueued for synchronization after the transaction is committed, wherein leaf object synchronization includes using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

32. (Currently Amended) A system for changing objects defining state of a computer program application, comprising:

means for creating a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

means for traversing a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

means for traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph, wherein the values for objects marked as dirty are recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

33-34. (Canceled)

35. (Currently Amended) A computer program product, tangibly stored on a computer-readable medium, for managing ~~maintaining~~ dependencies among a first set of objects each having a value, the first set of objects including ~~an object A and an~~ a first object [[B]] and one or more second objects depending directly or indirectly on the first object, the one or more second objects including at least one observer-only second object, the product comprising instructions operable to cause a computer to:

~~recompute the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, object A is marked as dirty and the value of object A is not recomputed until object A is queried for a value;~~

recompute the value of the first object [[B]], wherein when the value of the first object [[B]] changes, ~~the dependents of object B and all of their further dependents~~ the one or more second objects are invalidated, and the dependencies among the ~~dependents of object B and all of their further dependents~~ one or more second objects are severed, the value of each second object only being recomputed when the respective second object is queried for a value; and

cause each invalidated observer-only second object to recompute its value by querying the values of [[the]] a second set of objects from which the observer-only second object depends, wherein querying the values of the second set of objects causes invalidated objects in the second set of objects to be recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

36-37. (Canceled)

38. (Currently Amended) A computer program product, tangibly stored on a computer-readable medium, for changing objects having values defining state of a computer program application, the product comprising instructions operable to cause a computer to:

receive a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

register the change with a transaction;

dirty all objects dependent on the changed object;

sever dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueue the leaf object for synchronization after the transaction is committed, wherein leaf object synchronization includes using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

39. (Currently Amended) A computer program product, tangibly stored on a computer-readable medium, for changing objects defining state of a computer program application, the product comprising instructions operable to cause a computer to:

create a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

traverse a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traverse the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph, wherein the values for objects marked as dirty are recomputed using one or more requester objects, each requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

40-41. (Canceled)

42. (Currently Amended) The product of claim 35, further comprising instructions operable to:

provide the first object [[B]] in [[the]] a construction of each second object [[A]], wherein the value of each second object [[A]] is a function of the value of the first object [[B]] that was provided in the construction of the respective second object [[A]].

43. (Currently Amended) The product of claim 35, further comprising instructions operable to:

provide in the first object [[B]] a method that adds a requester owned by a second object [[A]] to a dependents list for the first object [[B]], the dependents list identifying all second objects whose value is a function of the value of the first object [[B]].

44. (Currently Amended) The product of claim 43, wherein the dependents lists for all objects in the first set of objects collectively define a directed, acyclic dependency graph.

45. (Currently Amended) The product of claim 35, further comprising instructions operable to:

break any dependency relationships the ~~marked~~ invalidated second object may have had when [[an]] a second object is invalidated ~~marked as dirty~~; and

when the value of an invalidated second object is recomputed, identify the objects on which the recomputed value is actually dependent and identify the recomputed second object as dependent only on the identified objects.

46. (Currently Amended) The product of claim 35, wherein the first set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid.

47-48. (Canceled)

49. (Previously Presented) The product of claim 38, wherein leaf object synchronization comprises:

recomputing a value for each object marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

50. (Canceled)

51. (Previously Presented) The product of claim 39, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

52. (Previously Presented) The product of claim 39, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

53. (Previously Presented) The product of claim 52, wherein:

the leaf objects are coupled directly to the user interface.

54-61. (Canceled)

62. (Currently Amended) The system of claim 28, further comprising:

means for providing the first object [[B]] in [[the]] a construction of each second object [[A]], wherein the value of each second object [[A]] is a function of the value of the first object [[B]] that was provided in the construction of the respective second object [[A]].

63. (Currently Amended) The system of claim 28, further comprising:

means for providing in the first object [[B]] a method that adds a requester owned by a second object [[A]] to a dependents list for the first object [[B]], the dependents list identifying all second objects whose value is a function of the value of the first object [[B]].

64. (Currently Amended) The system of claim 63, wherein the dependents lists for all objects in the first set of objects collectively define a directed, acyclic dependency graph.

65. (Currently Amended) The system of claim 28, further comprising:

means for breaking any dependency relationships the ~~marked~~ invalidated second object may have had when [[an]] a second object is invalidated ~~marked as dirty~~; and

when the value of an invalidated second object is recomputed, means for identifying the objects on which the recomputed value is actually dependent and means for identifying the recomputed second object as dependent only on the identified objects.

66. (Currently Amended) The system of claim 28, wherein the first set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid.

67-68. (Canceled)

69. (Previously Presented) The system of claim 31, further comprising:

for leaf object synchronization, means for recomputing a value for each object marked as dirty, means for identifying the objects on which the recomputed value is actually dependent, and means for identifying the recomputed object as dependent only on the identified objects.

70. (Canceled)

71. (Previously Presented) The system of claim 32, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

72. (Previously Presented) The system of claim 32, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

73. (Previously Presented) The system of claim 72, wherein:

the leaf objects are coupled directly to the user interface.

74-77. (Canceled)